PARALLEL PROCESSORS AND NONLINEAR STRUCTURAL DYNAMICS ALGORITHMS AND SOFTWARE

Principal Investigator: Ted Belytschko

Department of Civil Engineering
Northwestern University
Evanston, Illinois 60208-3109

Semiannual Progress Report

December 1, 1988 though May 31, 1989

NASA Research Grant NAG-1-650

PREFACE

This research was conducted under the direction of Professor Ted Belytschko. Participating research assistants were Noreen D. Gilbertsen, Mark O. Neal, and Edward J. Plaskacz. The help of Argonne National Laboratory, particularly Dr. James M. Kennedy who provided access to several parallel computing machines, is also appreciated.

The following papers supported by NASA-Langley were submitted for publication:

Ted Belytschko, Edward J. Plaskacz, James M. Kennedy, and Donald L. Greenwell, "Finite Element Analysis on the CONNECTION Machine", Computer Methods in Applied Mechanics and Engineering, submitted.

Mark O. Neal and Ted Belytschko, "Explicit-Explicit Subcycling with Non-Integer Time Step Ratios for Structural Dynamic Systems", Computers and Structures, submitted.

P. Smolinski, Ted Belytschko, and M. Neal, "Multi-Time-Step Integration Using Nodal Partitioning", International Journal for Numerical Methods in Engineering, 26, 349-359, 1988.

Abstract

This paper describes the adaptation of a finite element program with explicit time integration to a massively parallel SIMD computer, the CONNECTION Machine. The adaptation required the development of a new algorithm, called the exchange algorithm, in which all nodal variables are allocated to the element with an exchange of nodal forces at each time step. The architectural and C* programming language features of the CONNECTION Machine are also summarized. Various alternate data structures and associated algorithms for nonlinear finite element analysis are discussed and compared. Results are presented which demonstrate that the CONNECTION Machine is capable of outperforming the CRAY XMP/14.

1. INTRODUCTION

This paper describes a data management scheme and an associated algorithm for highly parallel (on the order of 10^3 processors) computers with local memory for nonlinear finite element analysis. The algorithms are either explicit or iterative implicit. The salient feature of this method is the allocation of all nodal variables to the element and an exchange of nodal forces at each time step in the procedure to maintain compatibility between elements. This algorithm has proven highly efficient on the CONNECTION machine. For a 16 K processor model, the speed of the CRAY X-MP/14 is exceeded by a factor of 3 for meshes consisting of 16 K elements. Even greater gains in speed are anticipated for larger meshes running on larger versions of the CONNECTION machine. The largest version of the CONNECTION machine has 65,536 (64 K) processors.

The development of the CONNECTION machine is a manifestation of aggressive competition to attain speedups in the computer design community.

It is widely acknowledged that the computational performance enhancements of a computer based on a Von Neumann architecture, i.e. single-instruction single-data (SISD) are approaching an asymptote due to the material limitations of the underlying hardware. Thus increasing attention has been devoted to alternate computer architectures falling under the three broad categories, first described by Flynn [6] in 1966 and summarized by Desrouchers [5]:

1. SIMD - single instruction multiple data. All processors execute the same instruction, however each processor uses its own data.
2. MISD - multiple instruction single data. Each processor has a unique instruction stream which operates on the same data stream.
3. MIMD - multiple instruction multiple data. Each processor has its own independent instruction and data streams. In general, processors are operating asynchronously and communication between processors is minimal.

Each of these alternate architectures requires a change in the way a programmer formulates his algorithm and develops the underlying code. The programmer must establish a sensitivity to the computer architecture he is working with to a degree unprecedented in the SISD era. An efficient and effective algorithm exploits the strengths of the underlying computer architecture while de-emphasizing its weaknesses.

The CONNECTION machine consists of up to 64 K processors, each processor is able to communicate with any other, hence the name "CONNECTION" machine. For massively parallel machines, the SIMD architecture with processor allocated memory is the most natural of the three alternate architectures. The programming model is to think of the processors as representing, for example, a collection of particles. Now given the task of calculating the state of each element, instead of using the SISD approach of looping over all

the particles one issues single instructions which are performed simultaneously by all the processors. This is an extremely natural way of performing the same set of instructions for each "element" in parallel.

While SIMD programming is initially easy to learn, to gain real performance requires experimentation. The general trends observed while implementing finite element algorithms on the CONNECTION Machine will be presented in subsequent sections.

A SIMD programming environment is reminiscent of vectorization with the exception that vectorization employs shared memory. Coarse grained SIMD parallelism with shared memory typically entails a distribution of data across a few processors in much the same way data is distributed across the registers of a vector processor. Shared memory simplifies the adaptation of algorithms designed for a Von Neumann architecture substantially. With a partitioned memory and fine grained parallelism the difficulties are much greater. Partitioned memory demands efficient data management which often results in a complete redesign of the algorithm. Where a sequential computer would have one processor do 8000 iterations of a loop, a coarse grained SIMD parallel (vector) computer with, for example, 8 processors (registers) will execute 1000 interations of the loop on each processor (register), and a fine grained (massively parallel) will execute the body of the loop once on each processor.

The differences in architecture also manifest themselves in the appearance of the final code. Code adaptation for a vector computer places a heavy emphasis on efficient GATHER-ASSEMBLE and eliminating nested inner loops by replication. This frequently obfuscates the underlying physics behind the computation. On the other hand, in a massively parallel partitioned memory environment state variables such as area are transformed from arrays in a SISD environment to scalars across the "width" of the computer. Inner loops remain

intact while outermost loops are replaced by a selection statement signaling that the subsequent body of code is to be executed in parallel. This inevitably results in a more readable code.

Parallel implementations have been considered by several investigators. Nour-Omid and Park [2] have reported on the implementation of implicit solvers on Hypercube MIMD machines with large partitioned memories. Belytschko and Gilbertsen [10] have described the implementation of explicit time integration with subcycling in a MIMD computer with shared memory. Neither of these implementations required the extent of restructuring of the algorithm and data structure as in a massively parallel SIMD partitioned architecture.

In this paper, the algorithm and data structure for dynamic nonlinear finite element analysis based on explicit time integration is described. A complete redesign which favors redundant calculations over the assembly of the global force vector from the element nodal force vectors (ASSEMBLE) was necessary. We will describe the redesigned algorithm and data structure in the context of one-dimensional and two-dimensional problems, although the three dimensional problem is most appropriate.

An outline of the paper is as follows. In Section 2, the CONNECTION machine architecture is described in greater detail. The focus is on hypercube topology and interprocessor communication. Section 3 describes the governing equations for finite element analysis and a comparison between their implementation on the Von Neumann and SIMD computers. Several CONNECTION Machine finite element algorithm prototypes are discussed. Section 4 describes the C* programming language for the CONNECTION Machine. Coding examples for the one dimensional problem are presented. Section 5 compares the performance of the CONNECTION Machine to the CRAY XMP/14 one of the most powerful SISD machines available. Section 6 summarizes and draws conclusions from all tests performed.

2. CONNECTION MACHINE ARCHITECTURE

The CONNECTION machine system consists of a front end computer, a parallel processing unit consisting of 16 K to 64 K data processors, each with a local memory, and (optionally) an I/O system that supports mass storage and graphics display devices. The front end computer is a conventional SISD computer which implements a standard operating system and extended versions of standard programming languages to facilitate code development. Programs developed for the CONNECTION machine are similar to programs developed for SISD machines with the exception that loops over the number of elements or the number of nodes in a finite element model are replaced by single commands activating many processors in the CONNECTION machine to simultaneously perform calculations on data residing within their local memories.

The parallel processing unit is an extension of the front end. One can think of the CM as intelligent memory. All code resides in the front end computer. The CM compliers translate serial code directly to the native assembly language of the front end while parallel code is translated to a mix of native assembly code and a special instruction set called PARIS (parallel instruction set). The PARIS calls result in operations addressing the front end bus interface which allows communication with the CONNECTION machine. In other words, upon encountering a parallel statement, the front end dictates the command to all activated processors. The fundamental building block of the parallel processing unit is an integrated circuit consisting of 16 processors and a routing device for interprocessor communication among processors located on different chips. Each processor has 64 K bits of memory. In addition, each pair of chips has an optional floating point accelerator. Taking advantage of the floating point accelerator requires no change in user software.

Each algorithm requires its own pattern of communication and hence each processor may need to communicate with any other. The construction of a direct connection between every pair of processors is impractical since a 64 K CONNECTION machine would require over 2 billion wires. Among the grid based processor interconnection schemes, the hypercube topology offers several distinct advantages. A single 16 processor chip can be thought of as a "zero-cube". Connecting two zero-cubes with a single wire yields a "one-cube". Connecting two "one-cubes" at their corresponding vertices yields a "two-cube". Repeating this process yields a "twelve-cube" with 2^{12} (4096) vertices. With 16 processors per vertex, a twelve cube can arrange 65,536 (64 K) processors with no processor more than 12 wires away.

It is important to note that in the description of the Boolean n-cube, each successive n-cube was established by linking the corresponding vertices of the previous cube. Therefore, each cube in an n-cube has two subcubes. A twelve cube is built up from two 11-cubes. Each 11-cube consists of two 10 cubes. This arrangement is in harmony with the binary logic of a computer as each subcube may be designated as either 0 or 1. Thus, each vertex of a 12-cube can be assigned a unique 12 bit address. Each message in the CONNECTION machine consists of an address and either data or an instruction. Messages are passed between routing devices which process the address field one bit at a time. There are many paths between any two processors; should some of them be blocked by other messages passing through the CONNECTION machine, the router may choose an alternate free path by simply processing the bits of the address in a different order.

Figure 1 illustrates two alternate routes between chips located on two vertices of a three-cube. The three-cube may be viewed as consisting of three pairs of planes. That is, a pair of x planes, a pair of y planes, and a pair

of z planes. Each plane within a pair is designated as 0 or 1. Thus each vertex of a three-cube can be assigned a unique binary address whose bits are determined by which member of each pair of planes it lies on.

In Figure 1a, a message is passed from 000 to 111. The router reads the first bit and sends the message to 100. There, the router reads the second bit and sends the message to 110. Finally the router reads the third bit and sends the message to its destination. Figure 1b illustrates the hypothetical situation where the router must select an alternate route between 000 and 111. At step one, the router has read the first bit of the destination address and has determined that the wire connecting 000 and 100 is already occupied by a transmission in progress. The router may then simply process the second bit first, sending the message to 010. The router then processes the first bit which results in the message being sent to 110. Finally the router processes the third bit and sends the message to is destination.

The same principle applies to the 12-cube of the CONNECTION Machine. Each vertex in the 12 cube has a unique address specified by a string of 12 bits. The first bit specifies which of the 11-cubes within the 12-cube contains the desired point. The second bit specifies which of the 10-cubes within the 11 cube contains the desired point and so on until the twelfth bit specifies which of the 0-cubes within the appropriate 1-cube is the desired vertex.

The above form of communication is called router communication and derives its name from the mechanism employed to pass messages between any two processors. A second mechanism for interprocessor communication which is much less general but, as will be subsequently shown, faster, is NEWS communication. The NEWS communication scheme establishes a PARIS instruction set through which the processors may be organized into an n-dimensional grid by

the programmer and every processor is allowed to send data to its immediate neighbors in the grid. n can be any integer between 1 and 31 (inclusive), and the size of a dimension must be a power of 2. NEWS interprocessor communication derives its name from the initials of the four principle directions: North, East, West, and South. This type of communication is only applicable to structured finite element meshes. The node spacing need not be regular but the element topology must be regular with at most 8 nearest neighbors. Meshes generated by quadtree and octree methods fit these criteria.

3. QUADRILATERAL PLANE STRESS CONTINUUM ELEMENT

A four node isoparametric quadrilateral element was implemented to design the algorithm and benchmark the performance of the CONNECTION Machine in the solution, for the displacement field $\underset{\sim}{u}(\underset{\sim}{x}, t)$, of the following initial value-boundary value problem.

equation of motion: $$\underset{\sim}{D}^T \underset{\sim}{\sigma}(\underset{\sim}{u}) + \underset{\sim}{b} = \rho \ddot{\underset{\sim}{u}} \quad (1)$$

strain-displacement: $$\underset{\sim}{\varepsilon} = \underset{\sim}{D}\,\underset{\sim}{u} \qquad \text{in } \Omega \quad (2)$$

stress-strain law: $$\underset{\sim}{\sigma} = \underset{\sim}{S}\,(\underset{\sim}{\varepsilon}) \quad (3)$$

boundary conditions

prescribed displacement: $$\underset{\sim}{u} = \bar{\underset{\sim}{u}} \text{ on } \Gamma_u \quad (4a)$$

prescribed traction: $$\underset{\sim}{\sigma}\,\underset{\sim}{n} = \bar{\underset{\sim}{\tau}} \text{ on } \Gamma_\tau \quad (4b)$$

$$\Gamma_u \cup \Gamma_\tau = \Gamma \quad (4c)$$

$$\Gamma_u \cap \Gamma_\tau = \phi \quad (4d)$$

initial conditions

initial displacement: $$\underset{\sim}{u}(\underset{\sim}{x}, 0) = \underset{\sim}{u}^0 \quad (5a)$$

initial velocity: $$\dot{\underset{\sim}{u}}(\underset{\sim}{x}, 0) = \dot{\underset{\sim}{u}}^0 \quad (5b)$$

where,

$$\underset{\sim}{u} = \begin{Bmatrix} u_x \\ u_y \end{Bmatrix} = \text{displacements, which are the unknowns} \tag{6}$$

$$\underset{\sim}{\varepsilon} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ 2\varepsilon_{xy} \end{Bmatrix} = \text{strains} \tag{7}$$

$$\underset{\sim}{\sigma} = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{Bmatrix} = \text{stresses} \tag{8}$$

$$\underset{\sim}{D} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \tag{9}$$

t denotes time

$\underset{\sim}{n}$ denotes a unit normal to Ω

A superimposed dot denotes differential with respect to time.

For a liner isotropic material, (3) is replaced by

$$\underset{\sim}{\sigma} = \underset{\sim}{C}\ \underset{\sim}{\varepsilon} \tag{10}$$

$$\underset{\sim}{C} = \frac{E}{1 - \nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{11}$$

$\underset{\sim}{C}$ is the constitutive matrix which for the case of isotropic linear elastic materials is determined solely by the two elastic constants E (Young's modulus or elastic modulus) and ν (Poisson's ratio).

In the finite element method the unknown, which is the displacement field $\underset{\sim}{u}\ (\underset{\sim}{x}, t)$, is interpolated by element shape functions $\underset{\sim}{N}$ in the form

$$\underset{\sim}{u}\ (\underset{\sim}{x}, t) = \underset{\sim}{N}_I\ (\underset{\sim}{x})\ \underset{\sim}{u}_{I_e}\ (t) = \underset{\sim}{N}\ \underset{\sim}{u}_e \tag{12}$$

where $\underset{\sim}{u}_e$ are the nodal displacements of the element. For the four-node quadrilateral which will be used here

$$\underset{\sim}{u}_e = \begin{Bmatrix} \underset{\sim}{u}_1 \\ \underset{\sim}{u}_2 \\ \underset{\sim}{u}_3 \\ \underset{\sim}{u}_4 \end{Bmatrix} \tag{13}$$

Each node of a 2D continuum element has two degrees of freedom.

$$u_I = \begin{Bmatrix} u_{xI} \\ u_{yI} \end{Bmatrix}$$

The relationship between the element nodal displacement vector and the global nodal displacement is given by

$$\underset{\sim}{u}_e = \underset{\sim}{L}_e\ \underset{\sim}{u} \tag{14}$$

where $\underset{\sim}{L}_e$ is a Boolean connectivity matrix for element e which maps global nodal quantities to local nodal quantities. This matrix is never actually constructed in finite element analysis. The information it contains is instead stored in an element connectivity array, IX(I,JE). IX(I,JE), I = 1,4 gives the global node numbers of element JE. The operation of extracting the

element nodal displacements from the global array, Eq. (14), is called the GATHER operation.

Performing a finite element semidiscretization on the initial value-boundary value problem, Eqs. (1) to (4) yields a system of ordinary differential equations in time (see Belytschko (1983)).

$$\underset{\sim}{M}_I \ddot{\underset{\sim}{u}}_I = \underset{\sim}{f}_I \qquad I = 1 \text{ to } n_n \tag{15}$$

where n_n is the number of nodes in the mesh. The nodal forces are given by

$$\underset{\sim}{f} = \sum_e \underset{\sim}{L}_e^T \underset{\sim}{f}_e \tag{16a}$$

$$\underset{\sim}{f}_e = \underset{\sim}{f}_e^{ext} - \underset{\sim}{f}_e^{int} \tag{16b}$$

$$\underset{\sim}{f}_e^{int} = \int_{\Omega_e} \underset{\sim}{B}^T \underset{\sim}{\sigma} \, d\Omega \tag{16c}$$

$$\underset{\sim}{f}_e^{ext} = \int_{\Omega_e} \underset{\sim}{N}^T \underset{\sim}{b} \, d\Omega + \int_{\Gamma_{\tau_e}} \underset{\sim}{N}^T \underset{\sim}{\tau}^* \, d\Omega \tag{16d}$$

$$\underset{\sim}{B} = \underset{\sim}{D} \, \underset{\sim}{N} \tag{16e}$$

$$\underset{\sim}{M}_e = \int_{\Omega_e} \rho \, \underset{\sim}{N}^T \underset{\sim}{N} \, d\Omega \tag{16f}$$

The mass matrix calculated in accordance with equation (16f) is called a consistent mass matrix. The lumped mass matrix is evaluated by adding all the terms in a row of the consistent mass matrix and placing the sum on the diagonal. Thus matrix inversion for the solution of (15) becomes trivial.

$$\underset{\sim}{M} = \sum_{e} \underset{\sim e}{L}^{T} \underset{\sim e}{M} \tag{16g}$$

We have written Eq. (15) in a nodal form, so that $\underset{\sim I}{f}$ are the nodal forces at node I. $\underset{\sim e}{f}$ is the nodal force matrix for an element, which in the case of a four-node quadrilateral is an 8 row column matrix.

The operation indicated by Eq. (16a) consists of using the IX array to add the nodal forces into their appropriate locations; this is called an ASSEMBLE operation.

The internal nodal forces, Eq. (16c), are evaluated by one point Gaussian quadrature with stabilization, see Belytschko, et al. [9]. The equations (15) are integrated in time by the central difference explicit method. As can be seen from Table 1, the Von Neumann algorithm is naturally partitioned into an element portion and a node portion. The allocation of data within arrays, as depicted in Figure 2a, reflects the partitioning of the algorithm: element-type arrays store the stresses, strains, and other state variables at all of the elements in sequence; whereas node-type arrays store the displacements, velocities, accelerations, and nodal state variables in sequence. The interrelationship between element and nodal variables is implemented by the GATHER and ASSEMBLE operations. The GATHER operation is not easily vectorized or implemented in a SIMD computer, see Flanagan [11]. These operations are controlled by the element-node data which determine the $\underset{\sim}{L}^{e}$ matrices in Eqs. 14 and 16g. The interrelationship of the data on a Von Neumann Machine is shown in Figure 2a. It is readily observed from Table 1 and Figure 2a that the nodal and element data are completely separate, and are only connected by the GATHER-ASSEMBLE operations.

Table 1. Flowchart for Explicit Integration in Von Neumann Computer

1. Initial conditions: $\underset{\sim}{u}(0) = \underset{\sim}{u}^0$; $\dot{\underset{\sim}{u}}\left(-\frac{\Delta t}{2}\right) = \dot{\underset{\sim}{u}}^0$

2. Loop over elements: e = 1 to n_e

   a. GATHER $\underset{\sim}{u}_e$ from $\underset{\sim}{u}$, Eq. (14)

   b. Evaluate strains: $\underset{\sim}{\varepsilon} = \underset{\sim}{B}\ \underset{\sim}{u}_e$

   c. Evaluate stress: $\underset{\sim}{\sigma} = \underset{\sim}{S}\ (\underset{\sim}{\varepsilon})$

   d. Compute internal and external forces, $\underset{\sim}{f}^e$ by Eq. (16c) and (16d)

   e. ASSEMBLE $\underset{\sim}{f}_e$ into $\underset{\sim}{f}$, Eq. (16a)

   end loop over elements

3. Loop over nodes: I = 1 to n

   a. $\ddot{\underset{\sim}{u}}_I = \underset{\sim}{M}_I^{-1}\ \underset{\sim}{f}_I$

   b. $\dot{\underset{\sim}{u}}_I^{j+1/2} = \dot{\underset{\sim}{u}}_I^{j-1/2} + \Delta t^{j+1/2}\ \ddot{\underset{\sim}{u}}_I$

   c. $\underset{\sim}{u}_I^{j+1} = \underset{\sim}{u}_I^{j} + \Delta t^{j}\ \dot{\underset{\sim}{u}}_I^{j+1/2}$

   end loop over nodes

4. $t \leftarrow t + \Delta t$; $j \leftarrow j + 1$; go to 2

The SIMD algorithm and associated data structures developed here are shown in Table 2 and Figure 2b, respectively. As can be seen, the GATHER-ASSEMBLE operations are absent. Instead, an EXCHANGE of information in the form of the nodal forces occurs at each time step. The nodal velocities and displacements are stored for each element and are integrated as part of the element calculation. This approach requires extra storage and computations because most nodes, for the four node quadrilateral, are shared by four elements; however, the reduction in communication makes this approach work best. Furthermore, this approach is naturally congruent with the architecture of CONNECTION type machines. In these machines, the storage per processor easily accommodates the extra nodal variables. Furthermore, it is most important to make the calculations in all processors identical. As will be subsequently shown, separate element and nodal data bases were quite ineffective.

Explicit time integration requires a sufficiently small time step to prevent numerical instabilities, which could render the results worthless, from occurring. A linearized analysis of the central difference method reveals that, for an undamped system, the stable time step bound is given by:

$$\Delta t \leq \frac{2}{\omega_{max}} \tag{17}$$

where ω_{max} is the highest frequency of the system. The frequency ω_{max} corresponds to the maximum eigenvalue of the equation:

$$[K]\{u\} = \omega^2[M]\{u\} \tag{18}$$

where [K] is the global stiffness matrix and [M] is the global mass matrix.

Table 2. Flowchart for Explicit Integration in SIMD Computer

1. Initial conditions: $\underset{\sim}{u}^0 = \underset{\sim}{u}(0)$; $\dot{\underset{\sim}{u}}^{-1/2} = \dot{\underset{\sim}{u}}(0)$;
   $j = t = 0$; initialize elements

2. Do elements computations in parallel: $e = 1$ to n_e

   a. Evaluate strain: $\underset{\sim}{\varepsilon} = \underset{\sim}{B}\ \underset{\sim}{u}_e$

   b. Evaluate stresses: $\underset{\sim}{\sigma} = \underset{\sim}{S}\ (\underset{\sim}{\varepsilon})$

   c. Compute element forces: $\underset{\sim}{f}_e$ by Eq. (16c) and (16d)

   EXCHANGE $\underset{\sim}{f}_e$

   d. $\ddot{\underset{\sim}{u}}_e = \underset{\sim}{M}^{-1}\ \underset{\sim}{f}_e$

   e. $\dot{\underset{\sim}{u}}_e = \dot{u}_e^{\,j-1/2} + \Delta t^{j-1/2}\ \ddot{\underset{\sim}{u}}_e$

   f. $\underset{\sim}{u}_e^{j+1} = \underset{\sim}{u}_e^{j} + \Delta t^{j}\ \dot{\underset{\sim}{u}}_e$

   end element computations

3. $t \leftarrow t + \Delta t$; $j \leftarrow j + 1$; go to 2

The task of determining the stable time step bound can be cast into a form more amenable to the massively parallel SIMD architecture of the CONNECTION Machine.

Rayleigh's theorem bounds the maximum frequency of the system by the maximum frequency among all the elements of the system (see Flanagan and Belytschko [13]).

$$\omega_{max} \leq \underset{\text{for all } e}{\text{Maximum}} \; (\omega_{max})_e \tag{19}$$

Furthermore, solution of the corresponding eigenvalue problem for each element is not necessary. For the 4 node quadrilateral, the element frequency can be estimated from (Flanagan and Belytschko [12])

$$(\omega_{max}^2)_e \leq \frac{4}{\rho A^2} (\lambda + 2\mu) \sum_{i=1}^{2} \sum_{I=1}^{4} B_{iI} B_{iI} \tag{20}$$

where

ρ = density

A = area of the four node quadrilateral

λ and μ are the Lame constants

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \quad \text{and} \quad \mu = \frac{E}{2(1+\nu)}$$

The determination of Δt in every step involves a sort. Equations (17)

and (20) are used to compute the stable time step for each element in the mesh. The minimum time step thus found is used in the time integration of the equation of motion.

C* provides reduction operators i.e. operators which read values from all parallel processors and deliver a combined result. Two examples of reduction operators in C* are:

$$< ? = \qquad \text{minimum} \tag{21}$$

$$> ? = \qquad \text{maximum} \tag{22}$$

Thus to find the smallest element time step within the domain plane-stress-quad and assign it to the mono (front end variable) DELT, one would write

$$\text{DELT} = (< ? = \text{delte}) \tag{23}$$

The recent innovations in computer architectures have resulted in a diverse range of tools designed to enable the programmer to exploit the strengths of the underlying hardware. These tools include specialized machine programming languages such as PARIS (Parallel Instruction Set) for the CONNECTION machine or "dialects" of standard programming languages such as FORTRAN and C i.e. CM FORTRAN and C* for the CONNECTION machine. It is not uncommon for programs in computational mechanics to approach several tens of thousands lines in length. It is not feasible to "translate" these codes into each dialect for every computer being constructed. However, the different architectures need to be tested via some standard benchmark. Therefore, the approach taken within this study has been to translate the computationally

intensive portion of a FORTRAN program for the transient analysis of continua into C* and linking it to a FORTRAN driving program which manages I/O. This approach is cognizant of the vast volume of FORTRAN code which has been built up over the past 30 years as well as the limitations of FORTRAN which impede its use on modern computer architectures.

A major shortcoming of FORTRAN is its lack of inherent data structures. FORTRAN was designed to strictly provide a facility for "formula translation". Although data structures can be established within FORTRAN using various array interrelationships the process is both cumbersome and error prone.

The C* programming language was chosen since it incorporates a data structure which leads to a very natural style of coding in a parallel environment. As will be subsequently shown the code written in C* is nearly identical to the C code which would be written for a single element problem. Thus, parallel programming in the C* programming language consists of parallel execution of a single element code.

The algorithm discussed previously emerged from a study of several algorithms on a one dimensional wave propagation problem. Schematic diagrams of the three algorithms tested on the one dimensional problem are shown in Figure 3. In finite element analysis, the most computationally intensive portion of the program is the calculation of element internal forces. The first algorithm tested performed element internal force calculations on the CONNECTION Machine, passed the calculated internal forces to the VAX 8250 front end where the accelerations were calculated and integrated to yield updated displacements which needed to be passed back to the CONNECTION Machine for the element internal force calculation of the subsequent time step. A small 20 element 150 time step rod problem took 54 secs. (wall clock time).

This approach suffered from two major shortcomings. First, the power of

the CONNECTION machine was not exploited in the integration of the accelerations. That is, the equations of motion were solved in a serial manner on the front end. This accounted for 34 secs. of solution time. Second, the communication bandwidth between the front end and the CONNECTION machine is very narrow, therefore adding the contribution of element internal forces into the global internal force vector stored on the front end accounted for 15 secs. of solution time.

The second algorithm tested sought to remedy these shortcomings by performing all calculations on the CONNECTION machine. A finite element data structure was established through a union of two domain types similar to a Von Neumann structure. The element internal forces were calculated within the element domains while the equations of motion were integrated in the nodal domains. This approach suffered from the partitioning of the CONNECTION Machine into two parts only one of which was active at any instance of time. Its value lies in the light it sheds on router interprocessor communication. The same benchmark problem required a solution time of 525 secs. Calculation of element internal forces required 392 secs. since the values of the displacements had to be obtained from the nodal domains.

The third approach was to establish a finite element data structure consisting of a single domain type. A processor within the CONNECTION Machine was assigned to each element and its nodes. Thus, the accelerations are integrated redundantly. However, the only interprocessor communication required is the passing of internal forces from each element of to its adjacent neighbors. The time required for the solution of the benchmark problem described previously was 30 secs. Upon closer examination of the solution time required for each section of the program, it was observed that solving the equation of motion required 12 secs. This was caused when a loop

index over the number of nodes in the element defaulted to a poly (CONNECTION Machine) variable. Thus, this index was stored on the stack of every processor and tested individually. The redeclaration of this loop index as a mono (front end) variable lowered the solution time for the entire benchmark problem to 17 secs.

4. C* PROGRAMMING LANGUAGE

A two node, elastic rod element is presented to illucidate C* coding on the CONNECTION Machine. The rod element was chosen instead of the quadrilateral for conciseness and clarity. The differential equation governing the motion of an elastic rod under the dynamic application of a load is analogous to the initial value - boundary value problem given by eqs. (1) - (5b). These matrix-vector equations reduce to the analogous scalar equation for the one dimensional case. Each node of the 1D rod element has one degree of freedom. Therefore,

$$\underset{\sim}{u} = u_x$$

$$\underset{\sim}{\varepsilon} = \varepsilon_x$$

$$\underset{\sim}{\sigma} = \sigma_x$$

$$\underset{\sim}{D} = \frac{\partial}{\partial x}$$

$$\underset{\sim}{C} = E$$

$$\underset{\sim}{B} = \frac{1}{L} [-1, +1]$$

where L is the length of the element. Likewise, equations (15) - (16g) reduce to scalar equations over the number of nodes in the mesh.

The C programming language allows a set of related variables to be grouped together by declaring them to be members of a structure. The C* programming language extends the concept of a structure by allowing functions to be associated members. The union of a structure and its associated functions is called a domain. An example of a domain and a member is presented in Figure 4. The structure for each rod element contains all of the material and geometric properties of the rod element and the kinematic, mass, and force properties of the two adjacent nodes. The first two variables within the rod element structure nei and nej are pointers to the two neighboring rod elements. That is, the values of nei and nej are addresses for the starting memory location of the structures containing the information for the two neighboring rod elements. In the function build_fint internal force contributions are passed to an element from its two adjacent elements through the use of pointers, a standard feature of the C programming language. The conditionals test whether or not each end of the rod element lies on the boundary of the rod. Rod elements whose end lies on the boundary of the rod have a pointer to the null domain and hence receive no internal force contribution from a neighboring rod element on that end of the rod. It is important to notice that the two neighboring rod elements can be processed within any two arbitrary processors within the CONNECTION Machine. The programmer employs router communication to pass values of mass and internal force to the appropriate processors. The pointers may be either calculated

through the examination of the finite element connectivity array, be input in the context of a neighboring element array NEIGHB, or generated in the case of consecutively numbered elements. For the purpose of this study, the NEIGHB array was input. Each rod element had two entries in the NEIGHB array corresponding to its two neighboring rod elements. Rod elements bordering on the boundary of the rod were assigned a value of zero in the corresponding entry of the NEIGHB array.

In dynamic finite element analysis using an explicit time integration, mesh numbering is not a critical issue as it becomes when a Newton-based implicit time integrator is implemented. While use of router based interprocessor communication, permits treatment of completely unstructured meshes, it is instructive to examine the gains in speedup possible with NEWS based interprocessor communication. NEWS based interprocessor communication suffers from the dual drawbacks of requiring a rigid element numbering scheme and the necessity of specifying a limit on the number of elements along each dimension of the mesh at compilation time. However, the implementation of NEWS type interprocessor communication results in a dramatic reduction in program execution time.

The modification of the rod domain for the implementation of the NEWS communication scheme presented in Figure 5. For the rod element, additional floating point variables are included as the receiving address for the mass and internal force contributions of the two adjacent elements. In addition, two "boolean" integer variables facti and factj are included to exclude extraneous values of internal force and mass from entering into the calculation through the ends of the rod.

Structures analogous to those implemented for the rod problem were established for the two dimensional continuum element. In the case of router

based interprocessor communication, the structure for each plane stress continuum element contains all of the material and geometric properties of the element, values of stress and strain at the gauss points and the values of the kinematic, mass, and force variables at the four adjacent nodes. Eight variables within the plane stress quadrilateral continuum element structure are pointers to the eight neighboring quadrilateral elements. In the function build_fint internal force contributions are passed to an element from its eight adjacent neighbors through the use of pointers. These pointers were established via the neighboring element array NEIGHB which was input. Each plane stress quadrilateral continuum element had eight entries in the NEIGHB array corresponding to its eight neighboring quadrilateral elements. Quadrilateral elements bordering on the boundary of the body were assigned a value of zero in the corresponding entry of the NEIGHB array. The eight pointers and the corresponding eight adjacent elements of an arbitrary element k within a 2D mesh of plane stress quadrilateral continuum elements is illustrated in Figure 6.

The mass and internal force contributions from the eight neighboring elements of element k are obtained through an eight stage exchange operation consisting of eight router messages. For example, the internal force contribution to nodes 0 and 3 of element k from the element whose address is contained by the pointer neleft is obtained through the following commands.

```
if (neleft != (domain-plane-stress-quad *) 0)
fint [3] += neleft -> finte[2]
fint [0] += neleft -> finte[1]
```

The modification of the quadrilateral domain for the implementation of the NEWS communication scheme is analogous to the modifications required for the rod domain. For the case of a two dimensional mesh, once NEWS interprocessor communication is established through PARIS calls, a structured mesh of m x n elements where m and n are powers of two specified by the programmer is established. The elements are allocated among the processors so that, for example, elements in mesh position (1,1) and (2,1) are located in adjacent processors. Also elements in mesh positions (1,1), i.e. first row and first column, and (1,2) are located in adjacent processors. Thus the NEIGHB array need only be boolean to delineate mesh boundaries. All communication takes the form of a simultaneous shift of data along one of the NEWS grid axes. For example, the PARIS instructions which are necessary to send the internal force contribution for nodes 0 and 3 from the left neighbor are:

```
CM_send_to_news_always_1L(&efint0,&finte[1],0,CM_upward,32)
CM_send_to_news_always_1L(&efint3,&finte[2],0,CM_upward,32)
```

The internal force assembly operation takes on a very simple and elegant form diagrammed in Figure 7. The mass and internal force contributions from the eight neighboring elements of element 5 are obtained through an exchange operation consisting of four shifts.

Stage 1: Right Shift

The internal force values for nodes 1 and 2 are sent to temporary variables efint0 and efint3 respectively to the right neighbor of each element.

Stage 2: Left Shift

The internal force values for nodes 0 and 3 are sent to temporary variables efint1 and efint2 respectively to the left neighbor of each element.

Update the internal force at each node

```
fint0 += efint0 * fact1
fint1 += efint1 * factr
fint2 += efint2 * factr
fint3 += efint3 * fact1
```

At this point it is instructive to note that the internal force contribution of elements 4 and 6 to element 5 have already been accounted for. Likewise, the internal force contribution of elements 7 and 9 to element 8 and the internal force contribution of elements 1 and 3 to element 2 have already been accounted for. Thus, the "exchange and assembly" of the element force vector for element 5 can be completed with two more shifts.

Stage 3: Downward Shift

The internal force values for nodes 0 and 1 are sent to temporary variables efint3 and efint2 respectively to the bottom neighbor of each element.

Stage 4: Upward Shift

The internal force values for nodes 2 and 3 are sent to temporary variables efint1 and efint0 respectively to the top neighbor of each element.

Update the internal force at each node

```
fint0 += efint0 * factd
fint1 += efint1 * factd
```

```
fint2 += efint2 * factu

fint3 += efint3 * factu
```

The distinctions between router and NEWS based interprocessor communication are the necessities of establishing a NEWS grid and the use of PARIS calls to control the exchange operation. Router based interprocessor communication can be used with unstructured meshes and depends on pointers which is a concept indigenous to the C programming language.

5. COMPARISONS OF THE CONNECTION MACHINE VS. THE CRAY

The issue of obtaining benchmark timings from the CONNECTION Machine is one surrounded by quite a bit of controversy. Although CPU time is the universal measure of system and program performance on scalar as well as vector/parallel computers, the CONNECTION Machine cannot be reliably benchmarked in this manner at the current time. Front end CPU time is a deceiving statistic since it does not account for time during which the CONNECTION Machine is processing instructions while the front end may be idle or, for example, working on the compilation of another user's program. Although CONNECTION Machine busy time can be measured, this timing does not account for front end CPU usage for the input/output phase of the program. With these considerations in mind, wall clock time was chosen as the benchmark timing indicator for measuring CONNECTION Machine performance. Wall clock time assures a measurement of an upper bound.

The following tables attempt to present a comparison between the performance of the CONNECTION Machine vs. the performance of the CRAY in finite element analysis. It should be stressed that the CONNECTION Machine is at a disadvantage to the CRAY in at least two respects. First, the times reported for the CRAY are CPU times which include none of the system overhead

reflected in a wall clock time. Second, the runs made on the CONNECTION machine include the overhead of first reading in the input data on the front end, mesh generation on the front end, and then transferring the nodal and element data into the CONNECTION Machine. This was necessary since most production finite element codes involve many thousand of lines devoted to I/O which cannot be adapted to the massively parallel architectures of machines like the CONNECTION Machine and most finite element meshes are irregular with elements of varying sizes which precludes the generation of geometric data for each element of the mesh simultaneously by the CONNECTION Machine.

Nevertheless, the CONNECTION Machine out performed the CRAY by a significant margin (on the order of 10-15 times faster) for very large problems. The fact that the CONNECTION Machine is a first generation massively parallel computer indicates exciting computational potential for latter generation massively parallel computers with faster processors, larger local memory at each node, and improved communication hardware.

CONNECTION Machine Timings

Elastic Wave Rod Problem - 15000 time steps wall clock times in seconds.

Number of Elements	VAX 8250 Front End		SUN 4 Front End
	Router	NEWS	NEWS
8000	1384.	1116.	235.
16000	1800.	1463.	293.
32000	2592.	2213.	620.
64000	4504.	3946.	2529.

CRAY X-MP/14 vs. CONNECTION Machine Benchmark:

SUN 4 Front End - NEWS Interprocessor Communication

64000 elements 95000 time steps 3450 secs.

5.67×10^{-7} secs/element·time step

(See Pie Chart 1.)

CRAY X-MP/14, fully vectorized FORTRAN code

2000 element 15000 time steps cft 1.15 compiler 51 secs. CPU

1.70×10^{-6} secs/element·time step

$$\frac{\text{CRAY}}{\text{CONNECTION}} = \frac{17}{5.67} \approx 3$$

Projected Wall Clock time for a 256000 element 95000 time step run on a 64K processor CONNECTION machine ≈ 3450 secs.

Potential $\quad \frac{\text{CRAY}}{\text{CONNECTION}} = \frac{17}{1.42} \approx 12$

CONNECTION Machine Timings

1 pt quadrature with hourglass control

Elastic Beam Bending Problem - 7500 time steps wall clock times in seconds

Number of Elements	VAX 8250 Front End		SUN 4 Front End
	Router	NEWS	NEWS
8192	4568.	2034.	563.
16384	6174.	2962.	652.

CRAY X-MP/14 vs. CONNECTION Machine Benchmark:

SUN 4 Front End - NEWS Interprocessor Communication

16384 elements 25000 time steps 1512 secs.

3.69×10^{-6} secs/element·time step

(See Pie Chart 2.)

CRAY X-MP/14 FORTRAN code

1000 elements 1000 time steps cft 77 compiler 56. secs. CPU

56×10^{-6} secs/element·time step

Estimated CPU time with vectorization

9.3×10^{-6} secs/element·time step

$$\frac{\text{CRAY}}{\text{CONNECTION}} = \frac{9.3}{3.7} \approx 2.5$$

Projected Wall Clock time for a 65536 element 25000 time step run on a 64K processor CONNECTION machine ≈ 1512 secs.

Potential $\frac{\text{CRAY}}{\text{CONNECTION}} \approx 10$

6. CONCLUSIONS

A data management scheme and an associated algorithm for explicit or iterative implicit finite element analysis on massively parallel partitioned memory SIMD computers, such as the CONNECTION Machine, has been presented. A study of CONNECTION Machine architecture and features of the C* programming language along with their implications on coding finite element analysis codes for use on the CONNECTION Machine has shown that a trade off between redundancy of calculations and interprocessor communication must be made to

attain peak performance. Two benchmark problems were studied to compare the performance of the CONNECTION Machine to the CRAY XMP/14. The CONNECTION Machine out performed the CRAY for very large problems. The architecture of the CONNECTION Machine is well suited to finite element analysis applications and the studies of CONNECTION Machine performance indicate exciting computational potential for latter generation massively parallel computers with faster processors, larger local memory at each node, and improved communication hardware.

7. ACKNOWLEDGEMENTS

Computations were performed on the 16K processor CONNECTION Machine at the Advanced Computing Research Facility, Mathematics and Computer Science Division, Argonne National Laboratory. The support of the Department of Educational Programs, Reactor Analysis and Safety Division, Computing and Telecommunications Division, and the Center for Energy Research Computations at Argonne National Laboratory is gratefully acknowledged. Support of NASA-Langley under grant NAG-1-650 for two of the authors (TB and EJP) is gratefully acknowledged.

8. REFERENCES

[1] Thinking Machines Corporation, The Connection Machine System Software Documentation Set, Version 5.0
Model CM-2 Technical Summary
CM Front-End Subsystems
CM Programming in C*
CM Parallel Instruction Set
Thinking Machines Corporation, 245 First Street, Cambridge, Massachusetts 02142-1214.

[2] B. Nour-Omid and K. C. Park, Solving Structural Mechanics Problems on the CALTECH Hypercube Machine, Computer Methods in Applied Mechanics and Engineering 61 (1987) 161-176, North-Holland.

[3] Russel J. Doty, FE Analysis: The Decathlon For Computers, Mechanical Engineering, November 1988, 62-68.

[4] W. Daniel Hillis, The CONNECTION Machine, Scientific American, Volume 256, Number 6, June 1987, 108-115, Scientific American Inc., 415 Madison Ave., New York, NY 10017.

[5] George R. Desrouchers, Principles of Parallel and Multiprocessing, Intertext Publications, Inc., McGraw-Hill Book Company, New York, 1987.

[6] M. J. Flynn, Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers, September 1972.

[7] D. L. Greenwell, R. K. Kalia, J. C. Patterson, P. D. Vashista, "Molecular Dynamics Algorithm on the CONNECTION Machine," International Journal of High-Speed Computing, World Scientific Publishing Company, to be published in 1989.

[8] T. Belytschko, "An Overview of Semidiscretization and Time Integration Procedures," in Computational Methods for Transient Analysis, ed. by T. Belytschko and T. J. R. Hughes, North Holland, Amsterdam, 1983, pp. 1-66.

[9] T. Belytschko, J. S-J Ong, W. K. Liu and J. M. Kennedy, "Hourglass Control in Linear and Nonlinear Problems," Computer Methods in Applied Mechanics and Engineering, 43, 1984, pp. 251-276.

[10] T. Belytschko and N. Gilbertsen, "Concurrent and Vectorized Mixed Time, Explicit Nonlinear Structural Dynamics Algorithm," in Parallel Computations and Their Impact on Mechanics, ed. by A. K. Noor, ASME, New York, 1987, p. 279-290.

[11] D. P. Flanagan and L. M. Taylor, "Structuring Data for Concurrent Vectorized Processing in a Transient Dynamics Finite Element Program," ibid, pp. 280-291.

[12] D. P. Flanagan and T. Belytschko, "Eigenvalues and Stable Time Steps for the Uniform Strain Hexahedron and Quadrilateral," Journal of Applied Mechanics, 51, 1984, pp. 35-40.

[13] D. P. Flanagan and T. Belytschko, "Simultaneous Relaxation in Structural Dynamics," Journal of the Engineering Mechanics Division, ASCE, 107, 1981, pp. 1039-1055.

[14] W. Daniel Hillis, The CONNECTION Machine, The MIT Press, Cambridge, Massachusetts, 1987.

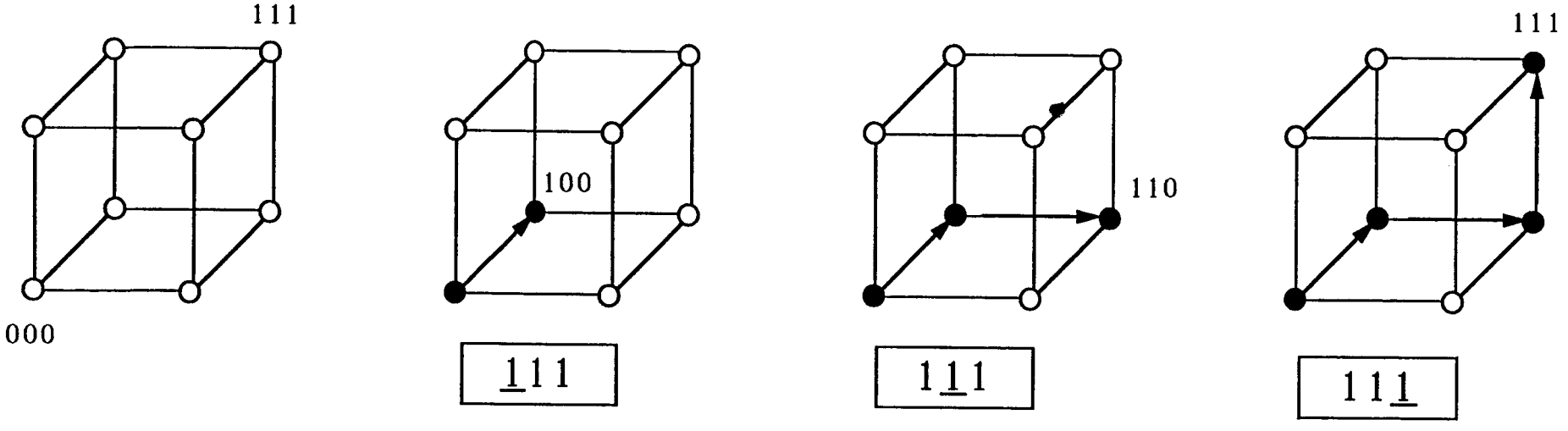


Figure 1a

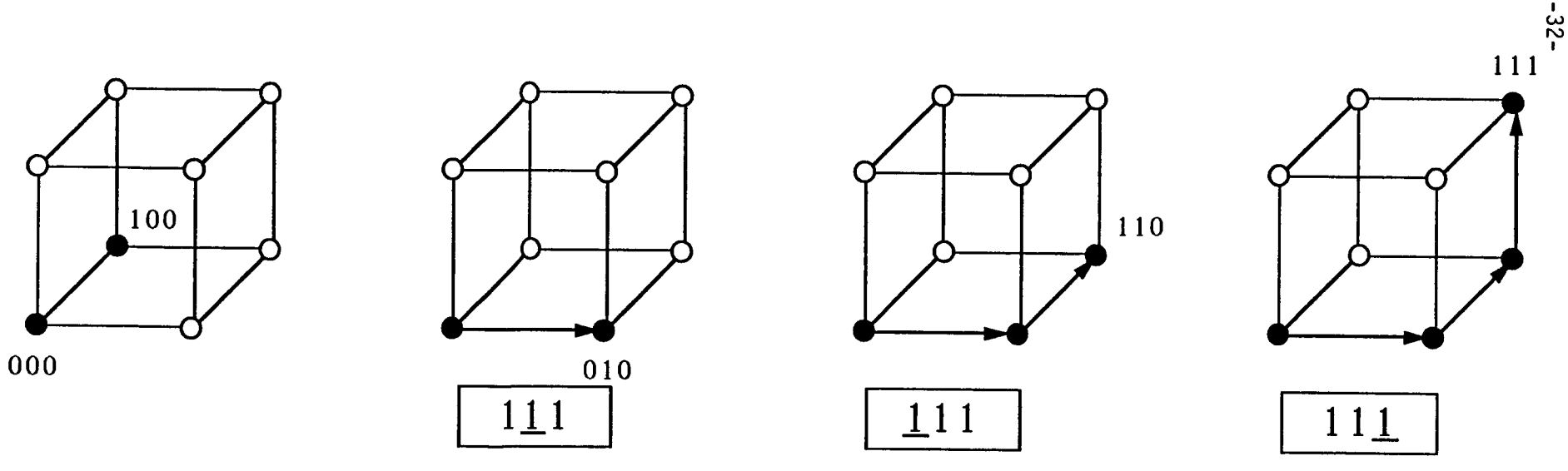


Figure 1b

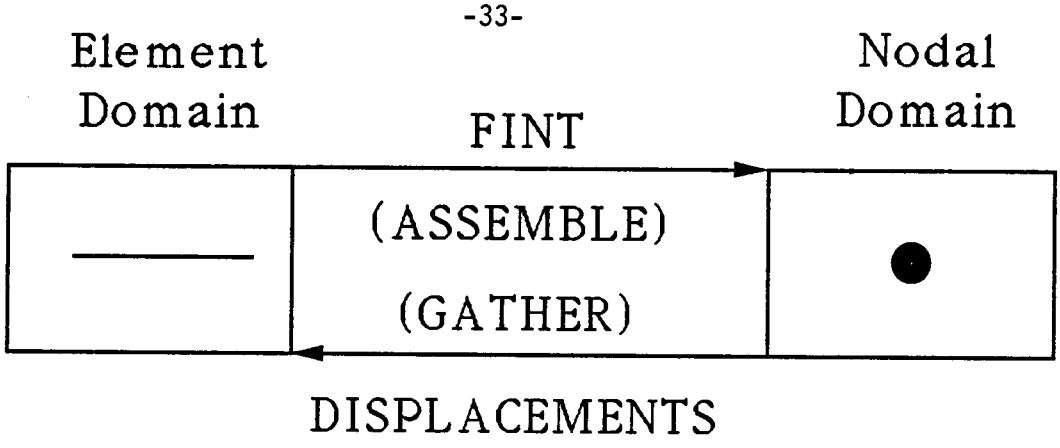


Figure 2a Von Neumann Data Structure

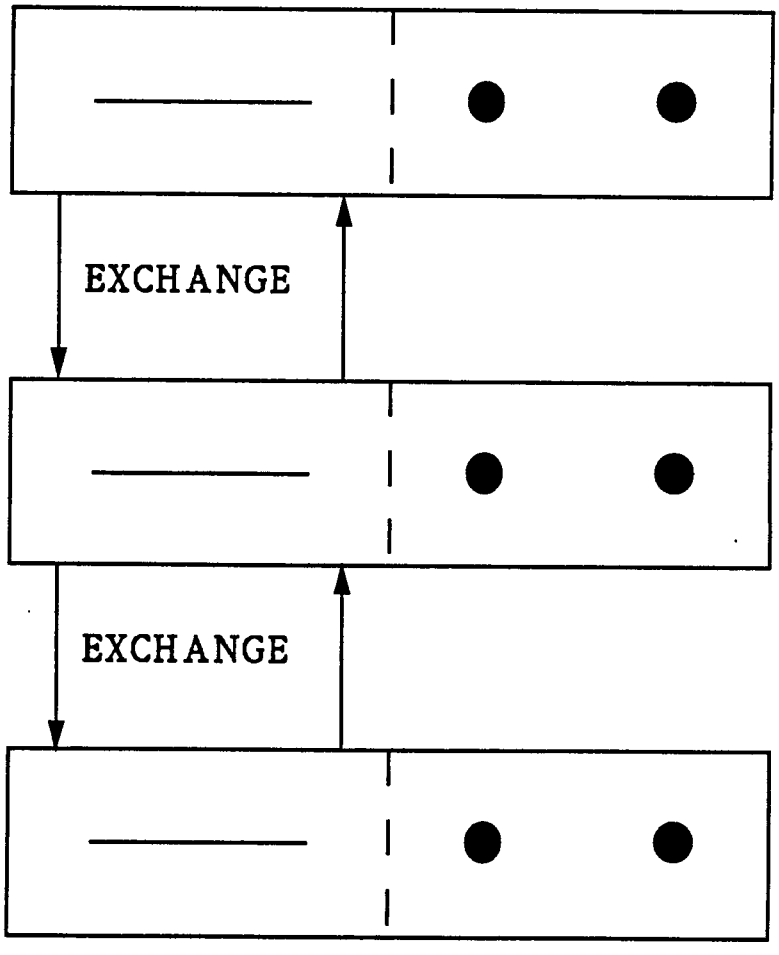
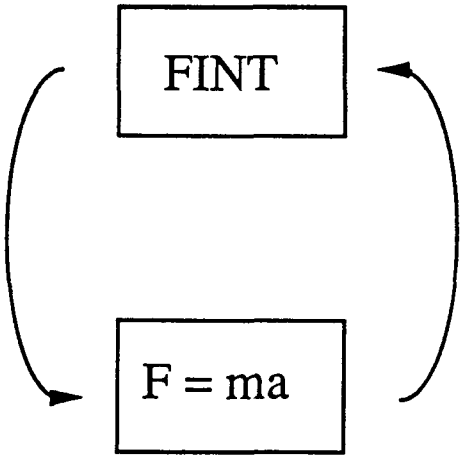


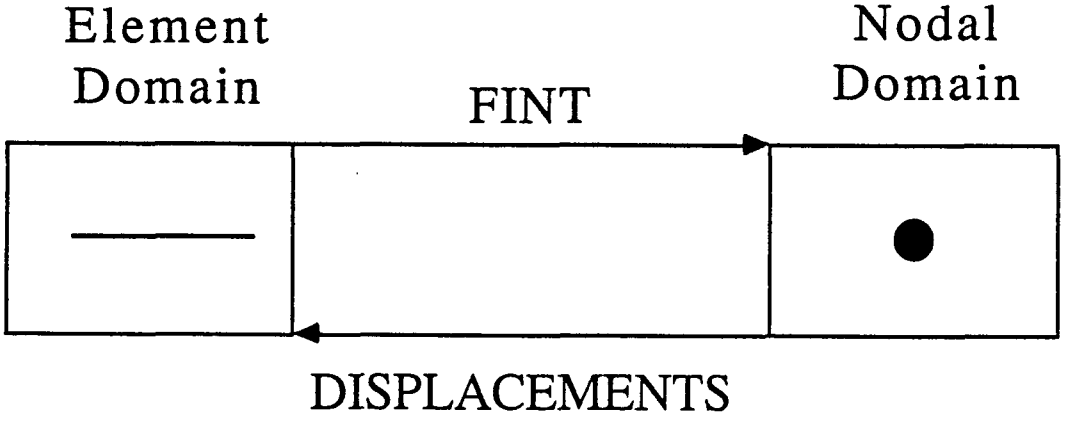
Figure 2b SIMD Data Structure

1. Calculation of Internal Force on the Connection Machine



Solution of Equation of Motion on Front End

2. Establishing a Finite Element Data Structure with Two Domain Types



3. Establish a Finite Element Data Structure with a Single Domain Type

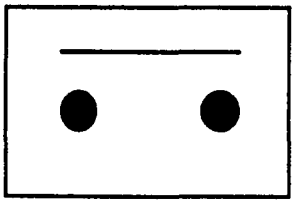


Figure 3

Example of a domain

```
domain  rod_element
{       domain  rod_element  *nei                                    ;
        domain  rod_element  *nej                                    ;
                int node[NPELE]                                      ;
                float  xlength                                       ;
                float  csarea                                        ;
                float  densty                                        ;
                float  emass                                         ;
                float  ym                                            ;
                float  stress                                        ;
                float  finte                                         ;
                float  delt                                          ;
                float coord[NPELE]                                   ;
                float disp[NPELE], vel[NPELE], acc[NPELE]            ;
                float fint[NPELE]                                    ;
                float fext[NPELE]                                    ;
                float mass[NPELE]                                    ;
                int bc[NPELE]                          ;       }     ;
```

Multiple instances of a domain via an array

```
domain  rod_element  element[MAXEL]                    ;
```

Examples of member functions

```
rod_element::calc_fint()
{       stress = (ym / xlength) * (disp[1] - disp[0])    ;
        finte = stress * area                            ;
        fint[0] = - finte                                ;
        fint[1] =   finte                                ;       }

rod_element::build_fint()
{       if (nej != (domain rod_element *) 0) fint[1] -= nej->finte ;
        if (nei != (domain rod_element *) 0) fint[0] += nei->finte ;  }
```

Activation of multiple processors is achieved through a selection statement

```
        [domain  rod_element].{calc_fint() ;}
```

Figure 4

Example of a domain

```
domain  rod_element
{           int indx                                          ;
            int index                                         ;
            int node[NPELE]                                   ;
            float xlength                                     ;
            float csarea                                      ;
            float densty                                      ;
            float emass, nei_emass, nej_emass                 ;
            float ym                                          ;
            float stress                                      ;
            float finte, nei_finte, nej_finte                 ;
            float delt                                        ;
            float coord[NPELE]                                ;
            float disp[NPELE], vel[NPELE], acc[NPELE]         ;
            float fint[NPELE]                                 ;
            float fext[NPELE]                                 ;
            float mass[NPELE]                                 ;
            int bc[NPELE]                                     ;
            int facti, factj                                  ;     };
```

Establishing a CONNECTION Machine geometry

```
rod_element  element::initialize_element_domain()
{
/*    Select all processors    */
          CM_set_context()                              ;
/*    All processors get their NEWS coordinate      */
          CM_my_news_coordinate_1L(&index,0,32)     ;
}
```

Example of NEWS communication

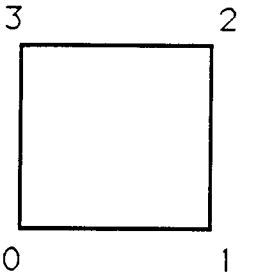
```
rod_element::build_fint()
{
/*    Each element receives the values of the element internal
      forces from its neightbors                                  */
       CM_send_to_news_always_1L(&nej_finte,&finte,0,CM_downward,32) ;
       CM_send_to_news_always_1L(&nei_finte,&finte,0,CM_upward,32) ;
      fint[0] += nei_finte * facti     ;
      fint[1] -= nej_finte * factj     ;
}
```

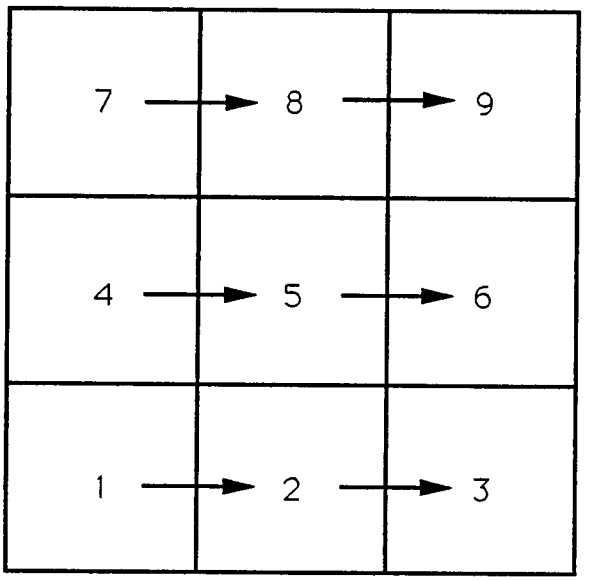
Figure 5

nediagupleft	neup	nediagupright
neleft	k	neright
nediagdownleft	nedown	nediagdownright

Figure 6

Stage 1:





Stage 2:

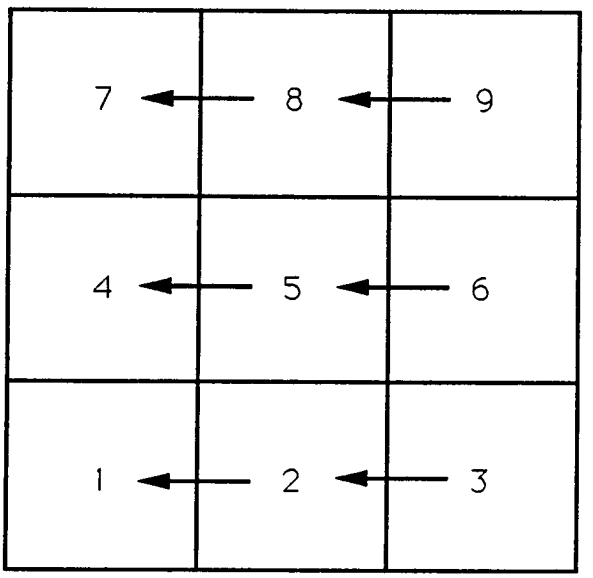
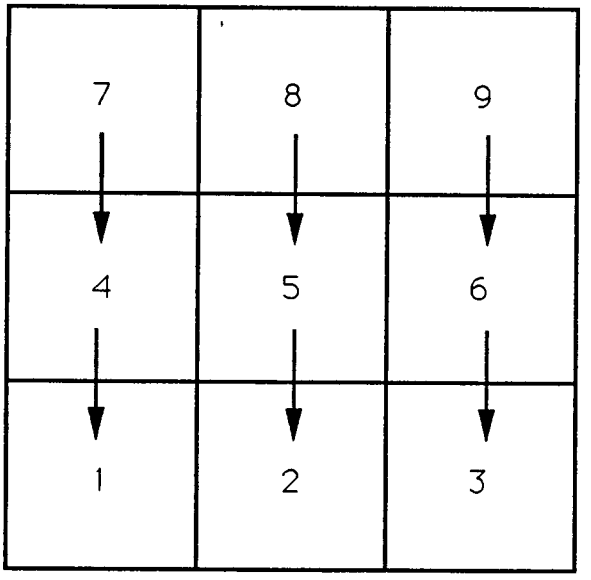


Figure 7

Stage 3:



Stage 4:

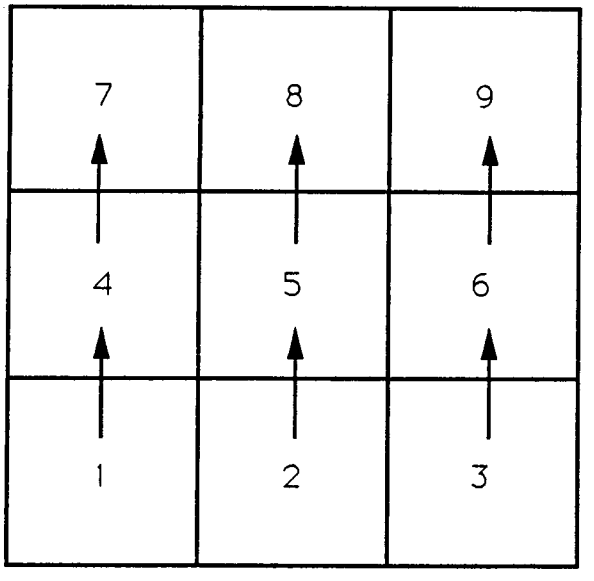
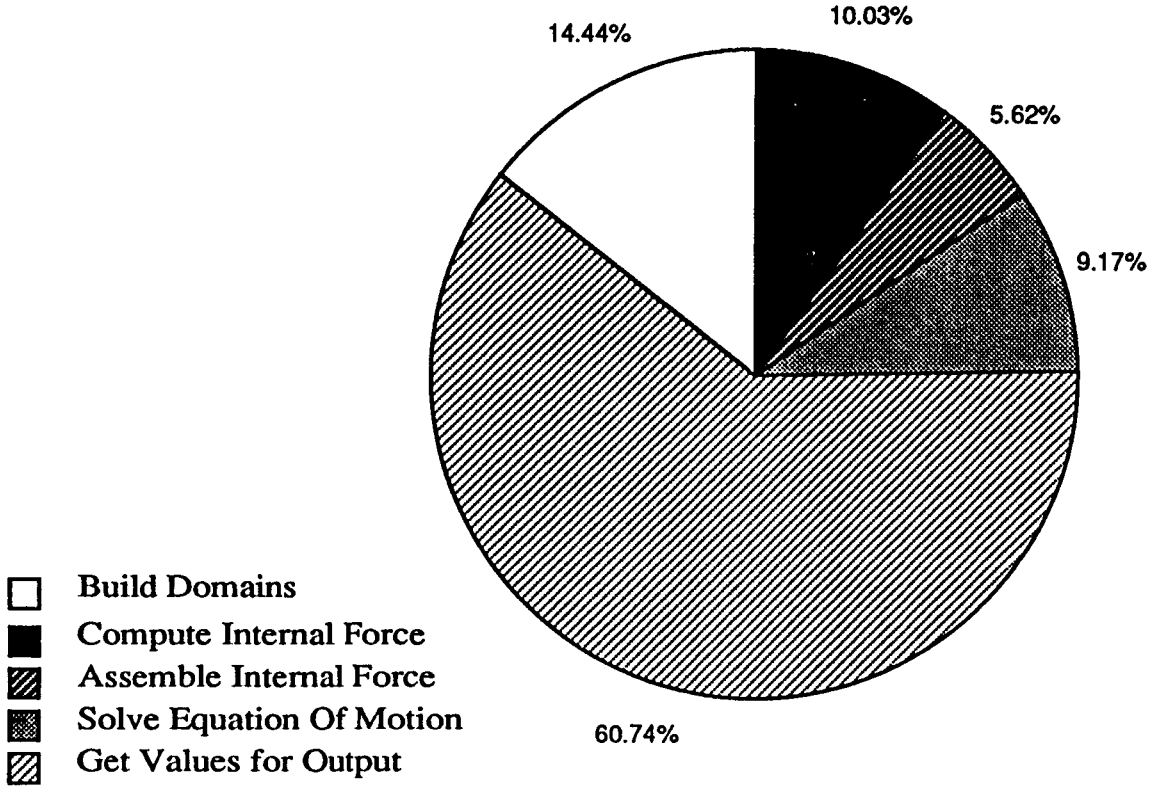


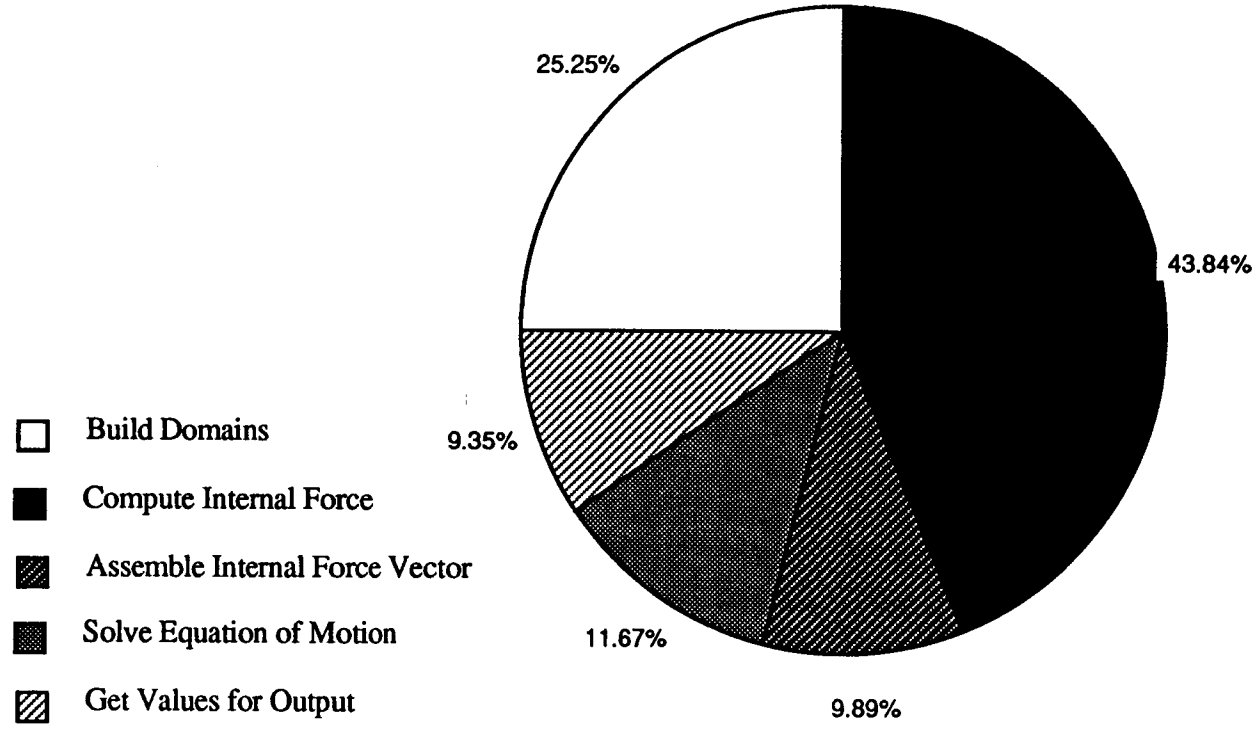
Figure 7

Timings For Elastic Rod -- 16000 Elements 95000 Time Steps
Wall Clock Time -- SUN 4 Front End



Pie Chart 1

Timings For Elastic Beam -- 16384 Elements 25000 Time Steps
Wall Clock Time -- SUN 4 Front End



Pie Chart 2